Electronic Version
Stylesheet Version v1.1.1

Description

Consuming Web Services on Demand

Background of Invention

Field of The Invention

[0001] The present invention is about consuming Web Services with instantaneous inputs, dynamic reconfiguration, transparent code generation, and multiple threads.

Related Art

[0002] Service Oriented Architecture (SOA) renders computer software as on-demand services. SOA is evolving into a pervasive enterprise computing platform because it affords more flexibility and efficiency in enterprise application integration than the traditional component architecture like EJB and COM.

[0003] Grid computing is a configuration, implementation, and exhibit of SOA. On-demand computing is the delivery of the computer grid. Web Services is the content and service delivered by on-demand computing under SOA, which comprises of following major characteristics: 1. Input to

the grid is instantaneous and spontaneous; 2. Grid wide parameters can be configured and reconfigured in real time; 3. The grid adopts a pay-as-you-go finance schedule.

[0004] Web Services Description Language (WSDL) is the cookbook for Web Services. Residing in the computer networks, WSDL is an XML file to describe Web Services. WSDL consists of two main parts, interface and implementation.

[0005] The interface part comprises of abstract descriptions of: 1. binding, specifying the network transport, such as SOAP, HTTP, SNMP, etc., to deliver described Web services; 2. portType, listing the operations of the described Web services; 3. types, defining the SOAP schema types of the input and output parameters for the operations.

[0006] Pointing to the actual service implementations, the implementation part contains one or more ports. Each port has an endpoint, network address where the described Web service resides. Each port points to a binding instance in the interface part.

[0007] Simple Object Access Protocol (SOAP) is stacked on top of HTTP and XML. HTTP is the network transport for SOAP while XML is the content format of SOAP messages. SOAP

is designed for remote method invocations by means of XML plain text messaging. SOAP is the most widely used messaging mechanism for Web Services.

[0008] Technologies bound for Web Services fall into three categories:

[0009] 1. Creating Web Services. It is a server technology that exposes existing or new software constructs as services by creating WSDL files for the services.

[0010] 2. Deploying and managing Web Services. It is a middle tier to server technology that involves UDDI registries and network configurations. See following US patent applications: a) Numbers 20030055878, 20030055868, and 20030055624 by James Fletcher et al; b) Numbers 20020178254, 20020178244, and 20020178214 by Peter Brittenham et al.

[0011] 3. Consuming Web Services. It is a client technology that invokes the services based on information provided by the WSDL files.

[0012] As the title has suggested, the present invention is about consuming Web Services. The US patent application titled "Invocation of Web Service from a Database" (number 20030093436 by Larry Brown et al) invokes Web Services with carefully constructed SQL queries. It's an attempt to

solve the client problem with a server side approach. The present invention invokes Web Services with client software constructs that are independent of database management systems. The US patent application titled "Testing Web Services as Components" (number 20030074423 by Thomas Mayberry et al) is an attempt to treat Web Services as software constructs under component architecture for testing purposes. The present invention itself follows SOA and takes a SOA approach throughout the Web Services testing process.

[0013] By adopting SOA and by making the process of consuming Web Services totally coding free from user's perspective, the present invention instantly making Web Services accessible via the Web and is readily configurable as a Web Services tester as well.

THE SYSTEM CONFIGURATION

[0014] A round trip of consuming Web Services involves steps of: 1. constructing SOAP request messages; 2. sending SOAP request messages via computer networks; 3. processing SOAP request messages; 4. invoking methods; 5. constructing SOAP response messages; 6. sending SOAP response messages via computer networks; 7. processing SOAP response messages.

[0015] Steps 1, 2, and 7 are client side operations; steps 3, 4, 5, and 6 server side operations. The present invention covers the client side operations that has a full spectrum of sub-operations. The present invention focuses on: interacting with end users with a Web browser; constructing invocation request objects from user data; making SOAP calls; and processing SOAP responses.

[0016] In the early stage of Internet, most of the Web pages are static HTML pages. Powered by the demand of dynamic content, Web page constructs like Microsoft's Active Server Page (ASP) and Sun Microsystems's Java Server Page (JSP) become the industry standards for dynamic pages. JSP is used by the computer program product because of its support for pure Java programming. The computer program product comprises of a set of JSP files that render HTML pages with real time data. The computer program product utilizes Apache Tomcat as the Servlet engine to run these JSP pages.

[0017] In the middle tier is a set of servlets deployed in Tomcat to direct the data traffic between the Web browser and the Java backend, which transforms user data like WSDL and concurrency configurations into Java objects. Meanwhile, the computer program product calls Axis api to generate

Java client stubs from WSDL such that network programming reduces to a local task. The computer program product constructs the invocation request object from user data and makes SOAP calls by invoking methods in the client stubs, which makes a client counterpart of the round trip server step 4.

[0018] When making SOAP calls the generated client stubs talk to Axis runtime, which serializes the Java objects into SOAP request messages and sends SOAP request messages over computer networks. Upon receipt of SOAP responses, AXIS deserializes them into Java objects and the computer program product processes the Java objects and present the result with JSP pages.

SUMMARY OF INVENTION

[0019] The present invention provides steps and means for on-demand coding-free service invocations by constructing invocation objects on the fly. The present invention is also completely end user coding free for multi-threading, for processing SOAP responses, and for measuring SOAP performance.

[0020] The conventional wisdom for on-demand invocations is to use DII Dynamic Invocation Interface, where steps of generating client stubs are skipped. While DII handles simple

SOAP data types at ease, it encounters great difficulties in constructing invocation inputs for complex data types. More over, manual coding is needed with DII in order to construct the invocation object. The present invention provides steps and means for automatically constructing invocation inputs with the generated client stubs up to an arbitrary complexity of input data types.

[0021] The present invention is configured as a Web Services tester when setting the number of invocation threads or the number of invocations per thread to be greater than one. The tester measures and presents SOAP performance in real time interaction with end users.

[0022] By adopting SOA, the present invention innovates in depth and width as embodied in: 1. cloning prior Web Services invocations; 2. processing overloaded methods; 3. dynamically reconfiguring Web Services by intercepting, transforming, and redirecting SOAP messages; 4. providing pay-as-you-go and prepaid finance schedules; 5. dynamically reconfiguring the list of operations to be invoked given a Web service; 6. instantly making Web Services accessible via the Web.

Brief Description of Drawings

[0023] *Fig. 1* depicts an *embodiment* of the present invention that

takes WSDL as an initial input and produces an invocation result as the output.

[0024] *Fig. 2* depicts a generic workflow of the present invention.

[0025] *Fig. 3* depicts the control flow of parsing a WSDL file.

[0026] *Fig. 4* depicts the control flow of constructing a Web Services invocation object.

[0027] *Fig. 5* depicts the control flow of constructing invocation inputs.

[0028] *Fig. 6* depicts the workflow of invoking a Web service with multiple threads.

[0029] *Fig. 7* depicts the control flow of cloning a prior Web Services invocation.

[0030] *Fig. 8* depicts the control flow of charging the end users.

DETAILED DESCRIPTION

[0031] The present invention is directed to a system, method, and computer program product for on-demand invocation of Web Services with multiple threads. The system functions as a computing grid in computer networks where it spontaneously processes concurrent Web Services invocation requests. For each invocation request, the system spurs one or more threads to invoke the Web service designated by the request. Then, the system presents the in-

vocation result including SOAP performance as automatically generated JSP pages.

An Overview of the Present Invention

[0032] The present invention hides all the complexity of invoking Web Services from end users. As depicted in *Fig. 1*, the initial input *60* to the system is the location of a WSDL file. Web Services invoker *100* retrieves WSDL from computer network *80*, transparently invokes Web Services described in the WSDL, and present the invocation result *180* via the computer network *160*. Input to the system is instantaneous and spontaneous. The system is interactive with end users, but under no circumstances manual coding of end users is needed for the present invention to carry out Web Services invocations.

[0033] *Fig. 2* offers a close view of how the present invention works in real world. First, the system presents a Web based form *200* to the end user. WSDL location is the required form field; concurrency configurations are optional from end user's perspective, which means the system assigns default values in the absence of user inputs. Second, the system simultaneously retrieves and parses the WSDL *210*, and saves the concurrency configurations *228*. Third, as a result of parsing, the system simultaneously gener-

ates Web based forms *220* and client stubs *226*. Fourth, the Web Services invoker *230* takes the form data, generated client stubs, and configurations and invokes the designated Web Services with multiple threads. Fifth, the system simultaneously processes SOAP responses *246* and measures SOAP performance *248*.

THE CONSTRUCTION PHASE

[0034] Prerequisites for invoking a Web service are: 1. Concurrency configurations that dictate the multi-threading behavior of the service invoker; 2. An invocation object that makes SOAP calls; 3. Invocation inputs that carry the parameter values of operations.

[0035] Concurrency configurations, either from user inputs or from system default values, consists of following entries: 1. The number of threads to be spurred for service invocation; 2. The initial number of threads to be spurred; 3. The time interval for the next thread upon the initial number of threads being spurred; 4. The number of repeated invocations for each invocation thread.

[0036] The present invention provides steps and means for transparently constructing the invocation object and inputs, which are preceded by parsing the WSDL file as depicted in *Fig. 3*. When the WSDL *300* is retrieved and vali-

dated in process *310*, the system parses the XML content of WSDL *328*. Then, the system runs two independent tests on SOAP endpoint *320* and port type *338*. If SOAP endpoint is found, the system generates client stubs *330*, and constructs invocation object *340*. If port type is found, the system generates Web based forms *348*.

[0037] Web based forms and client stubs are simultaneously generated for performance reasons. While end users are filling the forms with parameter values, the system is generating client stubs and constructing the invocation object in the background. When the form data is submitted and inputs are constructed, the system is ready to invoke the designated Web service and the end user will not experience noticeable latency in SOAP response.

[0038] *Fig. 4* provides a close view of how the invocation object is constructed. First, the system generates client stubs *408* from WSDL. Second, the system loads classes *418* from the generated stubs. Third, the system constructs the invocation object *420* from loaded classes. The system also offers shortcuts to the construction of invocation object provided that client stubs *400* and/or loaded classes *410* preexist. These shortcuts are designed for improving system performance because invocation object construction

is a resource intensive operation.

[0039] *Fig. 4* actually shows three routes to constructing an invocation object: 1. starting with WSDL, taking *400 408 418 420*; 2. starting with the generated stubs, taking *400 410 418 420*; 3. starting with the loaded classes, taking *400 410 420*.

[0040] *Fig. 5* extends process *348* in *Fig. 3* and provides a close view of constructing invocation inputs. The system parses the form data *500* and maps SOAP schema data types to those of underlying programming language. The computer program product aspect of the present invention maps SOAP types to Java types. The system runs a series of tests. The first one is an array test *510*, if true, recursively testing whether it"s an array of arrays. Otherwise, the system tests whether the SOAP type can be mapped to a basic type *520* for the underlying programming language; if true, mapping it to the basic type *528* and adding the parameter to the parameter list *548*. Otherwise, the system tries to use predefined typemappers *530* and *538*. Finally, the system introspects the generated client stubs *540* and maps to the types defined in the generated stubs.

[0041] *Fig. 5* depicts a parameter-by-parameter loop of constructing the invocation inputs. Here is the topology of

Web Services: a WSDL file describes one or more Web services; each Web service has one or more operations; and each operation consists of zero or more parameters. Therefore, *Fig. 5* is for a certain operation in a certain Web service described by a certain WSDL.

THE INVOCATION PHASE

[0042] The system packages the above-mentioned prerequisites into an invocation request object *610* as depicted in *Fig. 6*. Then the system submits the invocation request to the invocation queue *600*. A daemon thread called queue watcher *618* removes the request from the queue and spurs a client thread *628* for each request in the queue. Please note that the system processes concurrent submissions of WSDL and form data from end users. For each user submission, concurrency configurations are saved; invocation object and inputs constructed; and an invocation request submitted to the queue.

[0043] The client thread reads the concurrency configurations and further spurs one or more invocation threads *638*. Each invocation thread applies the invocation object and inputs, and invokes one or more times the designated Web service according to the concurrency configurations.

[0044] The present invention is a real time interactive system.

Upon submission of the Web based form *348*, the system manages the on-going Web Services invocation by pausing, stopping, or restarting the execution of invocation threads *638*.

[0045] The invocation result *630* consisting of outputs and SOAP performance measures is saved in real time while the invocation threads are executing. The outputs are SOAP responses, if successfully invoked, or SOAP fault messages otherwise.

[0046] When end users want to invoke a previously invoked Web service with the same set of inputs and concurrency configurations, instead of once again filling out the Web based forms *200*, *220*, and *500*, the system clones the prior Web Services invocation as depicted in *Fig. 7*.

[0047] As long as above-mentioned three prerequisites are copied over or reconstructed, the sequence of copying or reconstruction is not important. *Fig. 7* is an illustration of one of the many possible sequences. The system straightforwardly copies the concurrency configurations *710* upon receipt of the cloning request *700*. Then, it looks up the inputs *720* from cache. It copies inputs *730* or reconstructs it in process *726*.

[0048] Next, the system looks up the invocation object *740* in

cache, and copies *748* over if it's found. Otherwise, the system takes following three routes similar to those in *Fig. 4*: 1. starting with WSDL, taking *740 750 760 770*; 2. starting with the generated stubs, taking *740 750 760 766*; 3. starting with the loaded classes, taking *740 750 756*.

[0049] Again, the system uses the workflow in *Fig. 6* to invoke the previously invoked Web service and manages the cloned invocation by pausing, stopping, or restarting the execution of invocation threads *638*.

PROCESSING INVOCATION RESULTS

[0050] The present invention uses the open source program Apache Axis to process SOAP calls. Axis handles the deserialization of XML content in SOAP responses into Java objects. The present invention processes the Java objects and automatically generates JSP pages to display the invocation results with no need to code the visual presentation. The system processes an invocation result by constructing a table from the Java object. Each row in the table is a name value pair. The visual presentation is performed by fitting the table into a set of predefined templates that allow end users to tailor the report to their preferences.

[0051] The following WSDL fragment indicates the invocation re-

sult for operation GetStockQuotes is of type Array-OfQuote:

[0052] <s:element name="GetStockQuotesResponse"><s:complexType><s:sequence><s:element minOccurs="0" maxOccurs="1" name="GetStockQuotesResult" type="s0:ArrayOfQuote" /> </s:sequence> </s:complexType> </s:element>

[0053] The system retrieves quotes, say, for QQQ and SPY. Here is the SOAP response:

[0054] <?xml version="1.0" encoding="UTF-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GetStockQuotesResponse xmlns="http://swanandmokashi.com/"><GetStockQuotesResult><Quote><CompanyName>NASDAQ 100 TRUST</CompanyName><StockTicker>QQQ</StockTicker><StockQuote>31.80</StockQuote><LastUpdated>4:16pm</LastUpdated><Change>+0.38</Change><OpenPrice>31.79</OpenPrice><DayHighPrice>32.33</DayHigh-

Price><DayLowPrice>31.52</DayLowPrice><Volume>85
739504</Volume><MarketCap>N/A</MarketCap><Yea
rRange>19.76 -
32.75</YearRange></Quote><Quote><CompanyName
>S&P DEPOS
RECPTS</CompanyName><StockTicker>SPY</StockTicke
r><StockQuote>99.39</StockQuote><LastUpdated>4:1
5pm</LastUpdated><Change>+0.23</Change><OpenP
rice>99.98</OpenPrice><DayHighPrice>100.94</DayHi
gh-
Price><DayLowPrice>99.05</DayLowPrice><Volume>59
123300</Volume><MarketCap>N/A</MarketCap><Yea
rRange>77.07 -
102.179</YearRange></Quote></GetStockQuotesResult
></GetStockQuotesResponse></soap:Body></soap:Enve
lope>

[0055] Apache Axis constructs from the SOAP response a Java object of type ArrayOfQuote and the system constructs a table as listed below from the Java object. And the predefined templates provide a visual format of the table.

[0056] Stock Quotes Constructed from the ArrayOfQuote Java Object

CompanyName	NASDAQ 100 TRUST
StockTicker	QQQ

StockQuote	31.80
LastUpdated	4:16pm
Change	+0.38
OpenPrice	31.79
DayHighPrice	32.33
DayLowPrice	31.52
Volume	85739504
MarketCap	N/A
YearRange	19.76 - 32.75
CompanyName	S&P DEPOS RECPTS
StockTicker	SPY
StockQuote	99.39
LastUpdated	4:15pm
Change	+0.23
OpenPrice	99.98
DayHighPrice	100.94
DayLowPrice	99.05
Volume	59123300
MarketCap	N/A
YearRange	77.07 - 102.179

[0057] By the way, the SOAP request message is listed below:

[0058] <?xml version="1.0" encod-

ing="UTF-8"?><soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <soapenv:Body> <GetStockQuotes xmlns="http://swanandmokashi.com/"> <QuoteTicker>qqq,spy</QuoteTicker> </GetStockQuotes> </soapenv:Body></soapenv:Envelope>

[0059] In addition to presenting the invocation result as name value pairs, the present invention comprises of the computer program product to intercept and present raw SOAP request and response messages as listed above. This way, the computer program product has SOAP request messages transformed by end users, resent to the original SOAP endpoints, or a newly designated SOAP endpoints reconfigured in real time.

[0060] Apache Axis has a built-in TcpMon to do SOAP message transformation and redirection. It's a technology configured on the server side. The present invention is a client side technology innovation with no need to listen to any network communication ports. Also, TcpMon is a Java Swing program while the present invention is a Web based computer program product.

[0061] In case the SOAP response message carries attachments, the system first saves the attachments for each end user and categorizes the attachments by Web Services port name, then removes the attachments from system cache. In case of multi-threaded invocations, the system reconstructs the invocation object after each removal of the attachments from system cache.

[0062] The present invention is a system, method, and computer program product to take SOAP performance measures while invoking Web Services and processing SOAP responses. The system measures three parameters:

[0063] 1. SOAP throughput. It is the number of completed invocations in a given period of time, wherein a completed invocation is a SOAP request/response round trip. Each invocation request causes a client thread to be spurred, which in turn spurs one or more invocation threads according to the concurrency configurations. And each invocation thread performs one or more invocations as per the concurrency configurations. Therefore, SOAP throughput is a measurement for multiple transactions/invocations with multiple threads/virtual users. SOAP throughput measures the speed of the Web Services invoker and is an indicator in the Web Services performance testing.

[0064] 2. Active Invocation Threads. It is the number of virtual users that are in the process of invoking Web Services. The computer program product samples the data at a certain time interval and comes up an indication of the client load. The number of active invocation threads is an indicator in the Web Services load testing.

[0065] 3. Memory Usage. The computer program code measures the memory map in the heap allocation of the Java virtual machine and comes up with an indicator of how much memory is used by the virtual users/active invocation threads. Memory usage is an indicator in the Web Services performance testing.

[0066] The system uses all of the above-mentioned three parameters in benchmark testing Web Services, which consists of: 1. saving an invocation request as the baseline, and comparing performance and load of other invocation requests to the baseline; 2. saving at least two invocation requests as benchmarks, and comparing performance and load among the benchmarks.

[0067] The computer program product generates JSP pages and allows end users to configure and reconfigure baseline and benchmark comparisons in real time.

[0068] The computer program product computes SOAP through-

put by taking reciprocal of the moving average of SOAP response time. Meanwhile, the computer program product collects statistics of SOAP response time in real time: minimum response time, maximum response time, and mean response time.

[0069] The computer program product measures and displays the real time progress of Web Services invocations. The progress measurement has three indicators: percentage success, percentage fail, and percentage unfinished, which sums up to one hundred percent. The computer program product also measures and displays the elapsed time, and estimates time to completion.

THE OVERLOADED METHODS

[0070] Overloaded methods are methods of the same names but different method signatures. Method itself is a concept in object-oriented programming. In WSDL, the correspondent term is operation. Overloaded methods in WSDL are described as operations of the same name but different input/output parameters. There is a constant need to distinguish between overloaded methods from parsing WSDL to constructing invocation inputs. The challenge is that method name is presented in the Web based forms and the backend has to maintain and track the mapping be-

tween method names and the method objects. When methods are overloaded, the same method name can be mapped to different method objects.

[0071] The present invention provides a computer program product to process overloaded methods: 1. Overloaded methods in parsing WSDL *328* in*Fig. 3*. First, constructing operation objects from WSDL. Second, for each operation object, constructing a list of parameter types. Third, constructing a method table to map between the operation object and parameter type list; 2. Overloaded methods in constructing invocation inputs. First, listing method names in the Web based form *348* in*Fig. 3*. Second, associate each method name to an index that points to the position in the method table. Third, read the parameter type list from the method table. Fourth, constructing invocation inputs with the form data and the parameter type lists *500*, *510*, *520*, *530* in *Fig. 5*.

THE FINANCE SCHEDULES

[0072] The present invention provides two finance schedules for end users to use the system to invoke Web Services:

[0073] 1.*Pay-as-you-go* Schedule. When receiving new WSDL and concurrency configurations, the system presents a Web based form and demands payment for the invocation ser-

vice to be provided. The payment method is credit card, debit card, or check. The system processes the form data and seeks authorization of payment in real time. The system will not proceed to invoking the designated Web service until the payment authorization is successful.

[0074] 2.*Prepaid* Schedule. When receiving new WSDL and concurrency configurations, the system charges the end user out of the subscription plan where the end user has prepaid for the rights to use the system to invoke Web Services. The subscription plan carries a term limit and a virtual user limit.

[0075] The system is free of charge when the end user is of visitor status. The number of invocation thread, however, is limited to one for free invocation.

[0076] So far, the finance schedules are omitted in drawings for simplicity. A close look at *Fig. 2* gives an insertion point of the finance schedules between processes *200*, *210*, and *218*. Drilling down to the insertion point leads to *Fig. 8* where the system starts from processing form data *800* and ends with processing WSDL and saving concurrency configurations *848*. The finance schedules are depicted in the middle.

[0077] When receiving new WSDL and concurrency configura-

tions, the system first checks whether the end user has a subscription plan *810* to cover the Web Services invocation. If true, the system proceeds to process *848* directly. Otherwise, the system prompts the end user to subscribe the invocation service *820*. If the end user chooses to subscribe, the system presents a Web based subscription form *828*. The system presents a Web based payment form *830* whether the end user chooses to subscribe or not, as long as the system does not find a sufficient subscription plan to cover the Web Services invocation. The system processes payment form data and seeks a third party authorization *840* for the payment. The system proceeds to the next stage of processing WSDL and saving concurrency configurations *848* if the authorization is successful, and reports errors otherwise.

[0078] The pay-as-you-go finance schedule is embodied in the route of *800 810 820 830 840 848*. The prepaid finance schedule is embodied in two routes: *800 810 848*, and *800 810 820 828 830 840 848*. Both finance schedules charge on the basis of virtual users/number of invocation threads. The Web based payment form *830*, however, has different pricing for the two finance schedules. As the prepaid schedule also carries a term limit, pricing for the prepaid

schedule is based on the duration of the subscription and virtual users as well.

THE INVOCATION FORM

[0079] The Web based form depicted in boxes *220*, *348*, and *500* is an invocation form that allows end users to dynamically configure and reconfigure the list of operations to be invoked and to enter parameter values for each operation. The computer program product renders the invocation form in HTML with JSP.

[0080] The invocation form presents two lists of operations. One is the full list of operations included in the Web service. The other is the list of operations selected by end users for invoking the Web service. The second list is an invocation list and is a sub set of the first. The dynamically configured membership and the sequence in the invocation list dictate what operations are to be invoked and what the invocation sequence is.

[0081] When end users submit the invocation list along with the parameter values, they inadvertently double click the submit button and cause the invocation form to be submitted twice, which causes waste of network bandwidth and computer cycles in the backend. The present invention provides means for detecting and ignoring duplicate form

submissions by monitoring an order submission stack. Membership in the order stack indicates a duplicate submission, which is ignored unless it is a resend of a previously failed invocation.

[0082] The computer program product treats each new submission of WSDL *200* as a new order and saves concurrency configurations in the order object. The invocation form submission causes the concurrency configurations data to be transformed to the invocation request object and the order object to be persisted in the backend database.

PROGRAM LISTING DEPOSIT

[0083] /

****

********** LISTED BELOW ARE JAVA AND JSP CODE DISCLOSURE. JAVA EXCEPTION HANDLING CODE ARE STRIPPED OFF FOR SIMPLICITY.

****

**********/ /********** Claims 26, 27 **********/ private String parseServices(HttpSession session, Definition definition) { ..... if(endPoint == null || endPoint.compareTo("") == 0) parseError = "Can not find Soap endpoint in WSDL!"; else if(portName != null && portName.compareTo("") != 0) //

Generating client stubs spurCodeThread(morphID, wsdl, portName, orderNumber); ..... return portTypeName; // Then, generate Web based forms } private void spurCode-Thread(String morphID, String wsdl, String portName, String orderNumber) { Handle handle = new Han-dle(morphID, wsdl, portName, orderNumber); han-dle.setPriority(MAX_PRIORITY); handle.start(); } // Call Apache Axis api to generate client stubs in Java private void generate() { Emitter emitter = new Emitter(); emit-ter.setOutputDir(rootPath + File.separator + SOURCE_DIR); Parser parser = new Parser(); parser.setNowrap(true); parser = emitter; parser.run(wsdl); } / ********** Claims 26, 28, 29, 30, 31 **********/ // Construct from loaded classes public CodeGenerator(Object locator, String port-Name) throws InvocationException { this.locator = locator; this.portName = portName; } // Construct from generated client stubs public CodeGenerator(String portName, String orderNumber) { this.portName = portName; this.orderNumber = orderNumber; String cPath = DIR_GENERATED + File.separator + orderNumber + File.separator + CLASS_DIR; compiled = new File(cPath); locate(); } // Construct from WSDL public CodeGenera-

```
tor(String wsdl, String portName, String orderNumber) {
this.wsdl = wsdl; this.portName = portName;
this.orderNumber = orderNumber; rootPath =
DIR_GENERATED + File.separator + orderNumber; gener-
ate(); compile(); locate(); } public Object getInvokeHandle()
{ String methodName = "get" + portName; serviceObject
= clocator.getMethod(methodName, null).invoke(locator,
null); return serviceObject; } /
****************************** Claims 26, 28, 32
******************************/ public Object
soap2Java(String value, Class type) { if(value == null ||
type == Void.TYPE) return null; if(type ==
java.lang.String.class) return value; Object object = null;
if(type.isArray()) { object = getArray(value, type); } else {
object = tryFirstLayer(value, type); if(object == null) {
paramType = type; traverse(value, type, null); object =
parameter; } } return object; } /
****************************** Claims 26, 28, 33
******************************/ private void saveOrder() {
if(Form.isQuickOrder(session)) { order.setVirtualUsers(new
Integer(DEFAULT_VIRTUALUSERS)); order.setIterations(new
Integer(DEFAULT_ITERATIONS)); order.setInitialUsers(new
Integer(DEFAULT_VIRTUALUSERS)); or-
```

der.setUserStartInterval(new Integer(DEFAULT_INTERVAL)); } else { or- der.setVirtualUsers((Integer)session.getAttribute(VIRTUAL USERS)); or- der.setIterations((Integer)session.getAttribute(ITERATIONS)); or- der.setInitialUsers((Integer)session.getAttribute(INITIALUSE RS)); or- der.setUserStartInterval((Integer)session.getAttribute(INTE RVAL)); } } /****************************** Claims 24, 28, 34, 35, 36 ******************************/ public void placeOrder(HttpSession session, String morphID, boolean isResend) { String orderNumber = null; InvocationRequest r = new InvocationRequest(session, morphID); MorphServer.submit(r); } public InvocationRe- quest(HttpSession session, String morphID) { this.session = session; this.morphID = morphID; // methods carries the invocation inputs methods = (Hashtable)session.getAttribute(INVOKE_PARAMETER); or- derNumber = (String)session.getAttribute(ORDERNUMBER); // order car- ries the concurrency configurations Order order = new Order(morphID, orderNumber); // iresult carries the invo-

cation object Result iresult = new Result(orderNumber); } public synchronized void run() { while(true) { Invocation-Request request = (InvocationRequest)Queue.deque(); In-vocation invocation = new Invocation(request, orderNum-ber); Invocation.start(); } } public void invoke() { threads = new ThreadGroup(INVOKERS, orderNumber); for(int i = 0; i < virtualUsers; i++) { InvocationThread ithread = new In-vocationThread(new Integer(i).toString()); ithread.start(); } } /****************************** Claims 26, 37 ******************************/ public void pause(HttpServletRequest request, HttpServletResponse response) { MBean mbean = new MBean(); mbean.processBasic(request); Order order = mbean.getInvocationOrder(); Result iresult = mbean.getResult(); iresult.setOrderPaused(true); } public void stop(HttpServletRequest request, HttpServletRe-sponse response) { MBean mbean = new MBean(); mbean.processBasic(request); Order order = mbean.getInvocationOrder(); Result iresult = mbean.getResult(); iresult.setOrderPaused(false); ire-sult.setOrderCancelled(true); } public void restart(HttpServletRequest request, HttpServletResponse response) { MBean mbean = new MBean();

mbean.processBasic(request); Order order = mbean.getInvocationOrder(); Result iresult = mbean.getResult(); iresult.setOrderPaused(false); } public synchronized void run() { for(int k = 0; k < iterPerUser; k++) { while(iresult.orderPaused()) { try { Thread.currentThread().sleep(THREAD_INTERVAL); } catch(InterruptedException iex) {} } if(iresult.orderCancelled()) break; invokeService(); } } /********** Claims 26, 38 **********/ public static void invokeClone(HttpServletRequest request, HttpServletResponse response) throws ServletException { HttpSession session = request.getSession(); String orderNumber = null; MBean mbean = new MorphBean(); mbean.processBasic(request); Order ord = mbean.getInvocationOrder(); String ordNumber = ord.getOrderNumber(); String morphID = mbean.getUserName(); orderNumber = MorphData.dispatchOrderNumber(); InvocationRequest irequest = new InvocationRequest(morphID, ordNumber, orderNumber); MorphServer.submit(irequest); } // Constructor for cloning public InvocationRequest(String morphID, String ordNumber, String orderNumber) { this.morphID = morphID; this.ordNumber = ordNumber;

this.orderNumber = orderNumber; cloneIt(); locate(); } /
****************************** Claims 26, 39, 40
******************************/ public void savePerfor-
mance(HttpServletRequest request, HttpServletResponse
response) { String orderNumber = null; Result iresult =
null; MBean mbean = new MBean();
mbean.processBasic(request); Order order =
mbean.getInvocationOrder(); orderNumber = or-
der.getOrderNumber(); boolean baseline =
mbean.addAsBaseline(); boolean benchmark =
mbean.addAsBenchmark(); Port port = mbean.getPort();
iresult = mbean.getResult(); if(baseline && port != null) {
String previousBaseline = port.getBaseline();
port.setBaseline(orderNumber); String baselineName =
mbean.getBaselineName(); or-
der.setBaselineName(baselineName); String base-
lineDescription = mbean.getBaselineDescription(); or-
der.setBaselineDescription(baselineDescription); }
if(benchmark && port != null) {
port.addBenchmark(orderNumber); String benchmark-
Name = mbean.getBenchmarkName(); or-
der.setBenchmarkName(benchmarkName); String bench-
markDescription = mbean.getBenchmarkDescription(); or-

```
der.setBenchmarkDescription(benchmarkDescription); } }
private void invokeMethod(Method method, String
methodName, Object[] inputs, int iteration) { String iter =
"iteration " + iteration; Object result = null; String
soapRequest = null, soapResponse = null, fault = null;
long start = 0; long end = 0; try { start = Sys-
tem.currentTimeMillis(); result =
method.invoke(serviceObject, inputs); end = Sys-
tem.currentTimeMillis(); increment(threadIndex, 0); } catch
(Exception ex) { end = System.currentTimeMillis(); fault =
userPrefix + ", " + iter + "- Error occurred in invoking
method " + methodName + ".\n"; } long responseTime =
end - start; transactionCount++; totalTime += response-
Time; if(totalTime == 0) totalTime = 1; throughput =
1000*transactionCount/(int)totalTime; if(throughput ==
0) throughput = 1; if(fault != null && fault.compareTo("")
!= 0) increment(threadIndex, 1); mean += totalTime/
transactionCount; if(transactionCount == 1) { min = re-
sponseTime; max = responseTime; } else { if(min > re-
sponseTime) min = responseTime; if(max < response-
Time) max = responseTime; } } private void savePerfor-
mance() { int users = threads.activeCount(); invocationRe-
sult.setVirtualCount(users); performance[dIndex][0] = new
```

Long(System.currentTimeMillis()); performance[dIndex][1] = new Integer(users); performance[dIndex][2] = new Integer(throughput); performance[dIndex][3] = new Long((Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory())/1000); } public void drawStatus() { int successSum = 0, failSum = 0; int s = sTable.length; for(int i = 0; i < s; i++) { successSum += sTable[i][0].intValue(); failSum += sTable[i][1].intValue(); } int transactions = iresult.getTransactions().intValue(); int unfinished = transactions - successSum - failSum; DefaultPieDataset status = new DefaultPieDataset(); status.setValue("Fail", new Integer(failSum)); status.setValue("Success", new Integer(successSum)); status.setValue("Unfinished", new Integer(unfinished)); String title = "Success = " + successSum + " Fail = " + failSum + " Unfinished = " + unfinished; PieChart pieChart = new PieChart(CHART_WIDTH, CHART_HEIGHT, status, title); // The pie chart displays percentage success, percentage fail, // and percentage unfinished. } <% String altText = "Status Pie Chart"; String remark = ""; Plotter plotter = morpher.getPlotter(); plotter.drawStatus(); String startTime = new Date(order.getStartTime()).toString(); long pausedTime = 0; TreeSet set = order.listPausedTime(); if(set !=

null) { Iterator iterator = set.iterator();
while(iterator.hasNext()) { String pre =
(String)iterator.next(); pausedTime += new
Long(pre).longValue(); } } long elapsed = Sys-
tem.currentTimeMillis() - stime - pausedTime; long com-
pleted = morpher.getResult().getCompleted().longValue();
long estimatedTotal = 100*elapsed/completed; long esti-
matedLeft = estimatedTotal - elapsed; %> /
****************************** Claims 26, 41, 42
******************************/ public void
charge(HttpServletRequest request, boolean quickOrder)
throws ServletException { String error = null; HttpSession
session = request.getSession(); if(!quickOrder) // if not a
visitor request { String vUsers = re-
quest.getParameter(VIRTUALUSERS); virtualUsers = Inte-
ger.parseInt(vUsers); } if(isForSubscription(session)) error
= checkVirtualUsers(morphID, virtualUsers);
if(!isForSubscription(session)) charges =
PRICE_ONDEMAND*virtualUsers; if(authorize())
place(charges); } public String checkVirtualUsers(String
morphID, int virtualUsers) { String error = null; Customer
customer = new Customer(morphID); String transaction-
Number = customer.getPlanNumber(); Plan plan = new

```
Plan(customer, transactionNumber); int planVirtualUsers =
plan.getPlanVirtualUsers().intValue(); if(virtualUsers >
planVirtualUsers) return "The number of virtual users in
the order exceeds that in your subscription plan!"; TreeSet
set = new Morpher(morphID).listOrders(); Iterator iterator
= set.iterator(); int usersInuse = 0;
while(iterator.hasNext()) { String orderNumber =
(String)iterator.next(); Order order = new Order(morphID,
orderNumber); Boolean s = order.isForSubscription();
boolean forSubscription = false; if(s != null) forSubscrip-
tion = s.booleanValue(); String status = or-
der.getOrderStatus(); Integer v = order.getVirtualUsers();
int users = v.intValue(); usersInuse += users; } int to-
talusers = usersInuse + virtualUsers; if(totalusers > plan-
VirtualUsers) { error = "You ordered " + virtualUsers + "
virtualUsers, <br>momentarily, You have " +
(planVirtualUsers - usersInuse) + " left in the subscription
plan! To free up virtual users, " + "you can cancel open
orders or stop orders in execution."; } return error; } /
****************************** Claims 26, 43
******************************/ private void print(String
name, Class rType, Object result, Object[][] rt) { if(rType
== java.lang.String.class || rType.isPrimitive()) { if(rType
```

== java.lang.Void.TYPE) result = new String("void"); int i = getNameIndex(rt, name); if(rt[i][0] == null) rt[i][0] = name; String[] values = (String[])rt[i][1]; int index = 0; values = new String[PAGE_SIZE]; rt[i][1] = values; values[index] = result.toString().trim(); } else { Method[] methods = rType.getDeclaredMethods(); int count = methods.length; for(int i = 0; i < count; i++) { Method method = meth-ods[i]; String methodName = method.getName(); if(methodName.startsWith("get") && method.getModifiers() == 1 && method.getParameterTypes().length == 0) { Ob-ject object = method.invoke(result, null); processRe-turn(methodName.substring(3), object, rt); } } } } /********** Claims 26, 44, 45 **********/ public void save(Order or-der, Message sMessage, String methodName, Result ire-sult) { String orderNumber = order.getOrderNumber(); String morphID = order.getUserName(); Iterator iterator = sMessage.getAttachments(); int attachIndex = 0; File dir = null; while(iterator.hasNext()) { AttachmentPart attachment = (AttachmentPart)iterator.next(); InputStream fis = at-tachment.getDataHandler().getInputStream(); int size = fis.available(); byte[] bytes = new byte[size]; fis.read(bytes); fis.close(); String fileName = getFile-

Name(attachment, attachIndex); File file = new File(dir, fileName); file.createNewFile(); FileOutputStream fos = new FileOutputStream(file); fos.write(bytes); fos.close(); attachIndex++; } } public static void clear(Call call) { MessageContext context = call.getMessageContext(); Message sMessage = context.getResponseMessage(); Iterator iterator = sMessage.getAttachments(); while(iterator.hasNext()) { AttachmentPart att = (AttachmentPart)iterator.next(); String path = att.getDataHandler().getDataSource().getName(); File file = new File(path); file.delete(); att.clearContent(); att.removeAllMimeHeaders(); } } public synchronized void run() { for(int k = 0; k < iterPerUser; k++) { Set keys = methods.keySet(); Iterator iterator = keys.iterator(); while(iterator.hasNext()) { Method method = (Method)iterator.next(); String methodName = getMethodName(method, k); // this way can handle overloaded methods Object[] inputs = (Object[])methods.get(method); while(busy) try { wait(100); } catch(InterruptedException iex) {} busy = true; if(hasAttachments) refindMethod(methodName); invokeMethod(method, methodName, inputs, k); if(iresult.hasAttachments(methodName)) Attach-

ment.clear(locator); } // end while } // end for } /
****************************** Claims 26, 27, 45
******************************/ private void expandMeth-
ods(HttpSession session, List list, Definition definition) {
Hashtable methods = new Hashtable(); Hashtable
methodHints = new Hashtable(); Map mmap = defini-
tion.getMessages(); Set mset = mmap.keySet(); Iterator
miterator = mset.iterator(); while(miterator.hasNext()) {
Message message = (Message)mmap.get(miterator.next());
Operation method = getMethod(message, list, true);
if(method != null) { Hashtable parameters = new
Hashtable(); methods.put(method, parameters); Hashtable
pHints = new Hashtable(); // Use method object instead
of method name as key // to handle method overloading
methodHints.put(method, pHints); setParame-
ters(definition, message, parameters, pHints, true); } } }
<table> <% Set mset = methods.keySet(); Iterator mitera-
tor = mset.iterator(); int j = 0; while(miterator.hasNext()) {
Operation op = (Operation)miterator.next(); String mName
= op.getName(); if(j == index) { operation = op; method-
Name = mName; } j++; %> <tr> <td width="100%" >
<%= mName %></td> </tr> <% } %> </table> private
void refreshMethodList(HttpServletRequest request, Oper-

ation operation, Hashtable methods, Hashtable paramVal-
ues, Hashtable inputValues, Hashtable methodMap) {
Hashtable parameters =
(Hashtable)methods.get(operation); Set pset = parame-
ters.keySet(); Iterator piterator = pset.iterator(); int pcount
= pset.size(); Object []inputs = new Object[pcount];
Method method = getMethod(request, operation, parame-
ters); methodMap.put(operation, method); Class[] types =
method.getParameterTypes(); int pindex = 0;
while(piterator.hasNext()) { String parameterName =
(String)piterator.next(); String parameterType =
(String)parameters.get(parameterName); String value =
request.getParameter(parameterName); inputs[pindex] =
soap2Java(value, types[pindex]); pindex++; } paramVal-
ues.put(method, inputs); } /*****************************
Claims 26, 44, 46 *****************************/ private
void saveMessages(String methodName) { Call call =
(Call)locator.getClass().getMethod("getCall",
null).invoke(locator, null); MessageContext context =
call.getMessageContext(); Message rMessage = con-
text.getRequestMessage(); String soapRequest = rMes-
sage.getSOAPPartAsString(); Message sMessage = con-
text.getResponseMessage(); String soapResponse = sMes-

sage.getSOAPPartAsString(); ire-
sult.setSoapRequest(methodName, soapRequest); ire-
sult.setSoapResponse(methodName, soapResponse); } pri-
vate void redirect(String morphID, String orderNumber,
HttpServletRequest request) { String endPoint = null,
methodName = null; String soapRequest = null, soapRe-
sponse = null; Call call = null; Result iresult = null; Order
order = new Order(morphID, orderNumber); iresult = new
Result(orderNumber); iresult.setHandlerError("");
while(true) { // transformed SOAP request message
soapRequest = request.getParameter(REQUEST); method-
Name = request.getParameter(METHODNAME); endPoint =
request.getParameter(ENDPOINT); // dynamically recon-
figures SOAP endpoint order.setEndpoint(endPoint); break;
} Service service = new Service(); call =
(Call)service.createCall();
call.setTargetEndpointAddress(endPoint); byte[] sb =
soapRequest.getBytes(); ByteArrayInputStream bais = new
ByteArrayInputStream(sb); SOAPEnvelope envelope = new
SOAPEnvelope(bais); Message message = new Mes-
sage(envelope); call.setRequestMessage(message);
call.invoke(); Message responseMessage =
call.getMessageContext().getResponseMessage(); soapRe-

sponse = responseMessage.getSOAPPartAsString(); byte[] responseBytes = responseMessage.getSOAPPartAsBytes(); updateResult(new ByteArrayInputStream(responseBytes), iresult.getResult(methodName)); Attachment.save(order, responseMessage, methodName, iresult); if(iresult.hasAttachments(methodName)) Attachment.clear(call); iresult.setSoapRequest(methodName, soapRequest); iresult.setSoapResponse(methodName, soapResponse); } /****************************** Claims 26, 27, 45, 47 ******************************/ <TABLE WIDTH="100%" BORDER=0 CELLPADDING=5 cellspacing="0"> <% int size = 0; if(invocationList != null) { size = invocationList.size(); if(size > 1) { %> <TR> <TD width="35%" bgcolor="#C0C0C0" > Invocation List: </TD> <TD width="35%" bgcolor="#C0C0C0" > <% for(int k = 0; k < size; k++) { String boxName = Config.CHECKBOXNAME + k; int mindex = 0; Operation o = (Operation)invocationList.elementAt(k); String methName = o.getName(); int l = 0; Set kset = methods.keySet(); Iterator it = kset.iterator(); while(it.hasNext()) { Operation oper = (Operation)it.next(); if(oper.equals(o)) { mindex = l; break; } l++; } String target = ROOT + PAGE_FORM + "?" +

WSDL_METHODINDEX + "=" + mindex; %> <input type="checkbox" name="<%= boxName %>" value="<%= mindex %>" checked > <a href="<% out.print(target); %>"><%= methName %>
  <% } %> </TD> <TD width="30%" bg-color="#C0C0C0"> Uncheck a method to remove it from the Invocation List. The invocation se-quence is top-down on the list. </TD> </TR> <% } } %> <TR> <TD width="100%" colspan=3 height=18> </TD> </TR> <TR></TABLE> private void refreshInvocationList(HttpServletRequest request, Opera-tion operation, Hashtable methods, Hashtable paramVal-ues, Hashtable inputValues, boolean excluding, Hashtable methodMap) { HttpSession session = request.getSession(); Vector invocationList = (Vector)session.getAttribute(INVOKE_METHOD); if(invocationList == null) { invocationList = new Vector(); session.setAttribute(INVOKE_METHOD, invocationList); } Vector tempVector = (Vector)invocationList.clone(); int size = tempVector.size(); for(int i = 0; i < size; i++) { Op-eration oper = (Operation)tempVector.elementAt(i); String value = request.getParameter(CHECKBOXNAME + i); if(value == null || value.compareTo("") == 0) { invocation-

List.remove(oper); Method method = (Method)methodMap.get(oper); paramVal-ues.remove(method); inputValues.remove(method); } } if(!excluding && !invocationList.contains(operation)) invo-cationList.add(operation); } /****************************** Claims 26, 27, 48 ******************************/ public void buildInputs(HttpServletRequest request) { HttpSession session = request.getSession(); String morphID = check-Credential(session); String ordNumber = (String)session.getAttribute(ORDERNUMBER); Result iresult = new Result(ordNumber); boolean allow = false; if(orderVector.contains(ordNumber)) while(true) { Order order = new Order(null, ordNumber); TreeSet tset = or-der.listMethods(); Iterator iterator = tset.iterator(); while(iterator.hasNext()) { String methodName = (String)iterator.next(); String iError = ire-sult.getInvokeError(methodName); if(iError != null && iEr-ror.compareTo("") != 0) { allow = true; break; } } if(allow) break; return; } if(allow) iresult.setResendIndicator(true); buildInputs(); }